Normalization → to put all features values into a range so that we can find minima faster

features

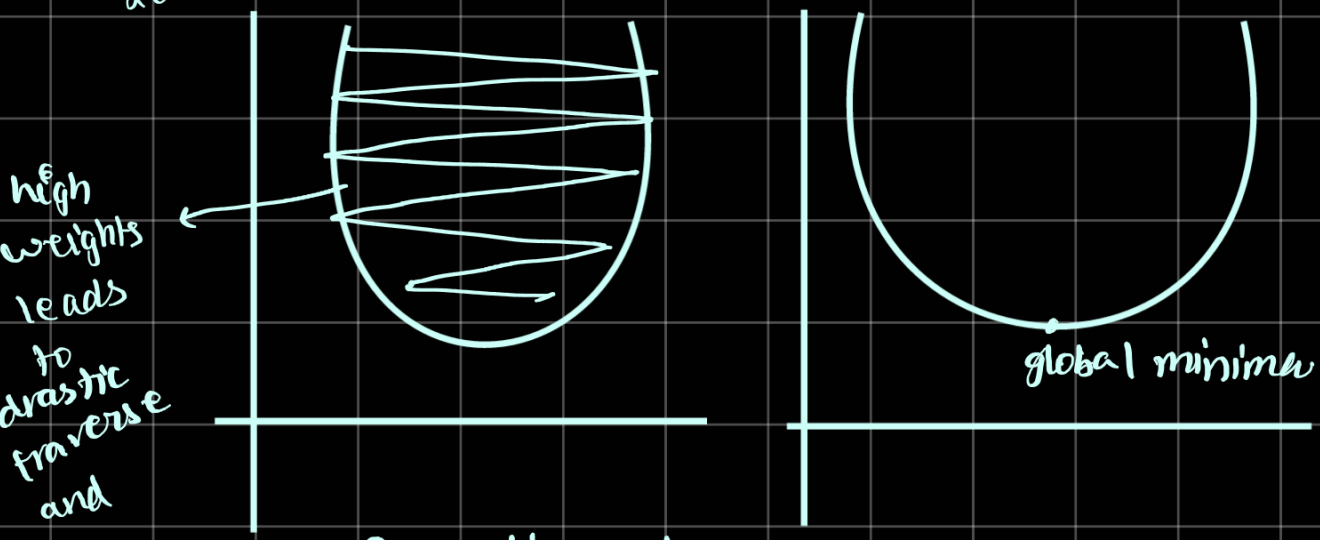
x_1	x_2
[-200, 200]	[-2, 2]
-200	02
0	0
-180	1
180	
200	

↓ Broad range

$w_1x_1 + w_2x_2 + \dots w_nx_n$

↓

for each range of x, the weight (θ) changes.



the probability of landing at global minima becomes low.

↳ Thus, we use normalization

$\frac{x_i - \text{mean}}{\text{standard deviation}}$] → weight updation would be uniform.

↓

Pytorch BatchNorm does the normalization.

↓

BatchNorm(2d)

↓

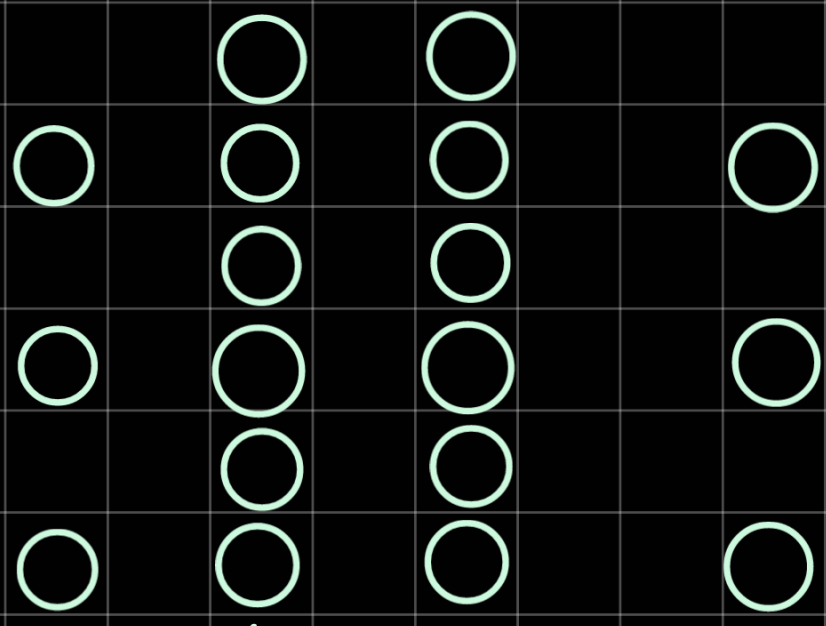
no. of dimensions

Coming out of Overfiting

method : Drop out

↓

few neurons are dropped ; no input



↳ some neurons are dropped during training (0.1% | 0.2% etc)

to inhibit aggressive learning and not deep learning.

↳ In the next iteration another batch of neurons are dropped randomly and previously inactive neurons will now get trained.

Note :- Drop-out technique can be using only in training but not in testing

x_1	x_2	y
1	2	6
2	3	7
1	4	10
1	5	7

we get appropriate weights when model is overfitting and we need probabilistic weights for generalization. So we penalize the model

$$y = ax_1 + x_2 + 1$$

(a: w_1, 1: w_2, 1: bias)

↓

Regularization

$$L = \frac{1}{m}\sum_{i=1}^{m}(\hat{y} - y) + \alpha \sum (w_i)$$

(Lasso regularization)

↑ if absolute values are subtracted: L_1 regularization

square " " " : L_2 "

↳ Ridge "

$$L = \frac{1}{m}\sum_{i=1}^{m}(\hat{y} - y) + \alpha \sum (w_i)^2$$

for scholastic gradient descent

hyper parameter (manual written)

SGD (m, lr, weight decay = 0.0001)

↳ does L_2 regularization

code for regularization

```
q = 0
for param in model.parameter():
        sum = sum + param
sum *= const
l += sum.
```

Early stopping

↳ If the epoch o/ps are getting saturated we use early stopping to fine tune till saturation point.